Collaboration Entities on Deterministic Finite Automata

Conference Paper · June 2006

DOI: 10.1109/CTS.2006.26 · Source: IEEE Xplore

3 authors, including:

Geoffrey Charles Fox
University of Virginia
1,515 PUBLICATIONS **29,078** CITATIONS

SEE PROFILE

Maisie Pierce
Nottingham Trent University
50 PUBLICATIONS **303** CITATIONS

SEE PROFILE

Collaboration Entities on Deterministic Finite Automata

Minjun Wang
EECS Department, Syracuse University, U.S.A
Community Grid Lab, Indiana University, U.S.A
501 N Morton, Suite 222, Bloomington IN 47404
minwang@indiana.edu

Geoffrey Fox
Community Grid Laboratory, Computer Science Department, School of Informatics and Physics Department, Indiana University, U.S.A
gcf@indiana.edu

Marlon Pierce
Community Grid Laboratory, Indiana University, U.S.A
501 N Morton, Suite 224, Bloomington IN 47404
mpierce@cs.indiana.edu

ABSTRACT

We have developed several types of collaborative applications. They are realizations of the Shared Event Model in Grid-base Collaboration, and examples of Peer-to-Peer Grid computing. Each application type consists of collaboration entities, and they play different roles in collaboration. These entities are finite automaton-based in a collaboration session; in essence, they are just deterministic finite automata in the session. Intuitively, the entities in collaboration collaborate on events to keep showing the same output displays at each step, with one entity in controlling by capturing events, generating and sending out event messages to the others through a message broker, and the others in responding by rendering the received event messages. Specifically, they collaborate to share a common finite automaton in their respective instantiations, and reach a common state of the finite automaton at any collaboration step. Collaboration of the entities is therefore all about being in a same state of the finite automaton at each event.

KEYWORDS: Collaboration, Event, Automaton, P2P, Grid.

1. INTRODUCTION

We have developed Collaborative PowerPoint applications [1], Collaborative Impress applications [2], and Collaborative ReviewPlus applications [3]. They are instantiations of Shared Event Model [4, 2] in Grid-base Collaboration, and can be used in e-Learning, distance education, online conference, e-Science, and more.

They work on a Grid-based Collaboration Paradigm [4], in which Shared Event Model as messenger, and Peer-to-Peer Grid computing [5, 6, and 1] as basis.

They are desktop windows collaborative applications. They are based on the stand-alone PowerPoint, Impress and ReviewPlus [7] applications, and are developed as collaborative applications that enable the otherwise stand-alone ones of each type to collaborate within themselves over the networks.

Impress of OpenOffice [8] is a presentation application similar to Microsoft PowerPoint, and has similar functionality. ReviewPlus is a general-purpose data visualization tool developed in Interactive Data Language (IDL) by General Atomics of USA. It is used in physics and engineering for displaying 2D and 3D graphs and signals.

We design the overall structure of each of the three collaborative applications to consist of a type of Master (or Master Client) and a type of Participant (or Participating Client) using small text event messages for the communication between them. During a session, the Master captures events in its process, deals with them, generates delimited event messages, and sends the event messages via a message broker to the participant for rendering the displays in the participant's space, so that both of them can share the screen displays simultaneously.

The Master is in active mode and controls the process of a session; the Participant is in passive mode and is not in control of the process; it just receives the event messages and renders the displays. There can be multiple Participants working with the Master concurrently and independently.

The collaboration is on the Shared Event Model; small size text event messages are communicated between the Master and Participants to synchronize their displays. Compared to other approaches of achieving synchronized views such as Shared Display (that communicates image data like bitmap) in Virtual Network Computing [9], this method uses small network traffic.

We use a common message broker – NaradaBrokering (NB) Message Service [10, 11] – as the underlying message communication system between the Master and Participant clients. It is deployed in Grid as a Grid service, and the clients are deployed on user computers and are running as Peers using the service for message communication, so that together they perform Peer-to-Peer Grid computing.

The base software – Microsoft Office, OpenOffice/Star Office, or RSI IDL – is required to install on both the hosts of the Master and the Participant, and if files are needed in a session, they are deployed beforehand on the same directories on the hosts. This deployment guarantees the accesses of the files are correct on the hosts under the control of event messages.

All clients are required to be in a session and keep in that session for the whole collaboration, because an event message coordinates each client to change its current status, and the correct transition to a subsequent status depends on the previous one.

In a collaboration session, we can generalize the collaborative applications on PowerPoint, Impress and ReviewPlus to "*Deterministic Finite Automaton-based Collaboration Entities*."

In the session, we can think of the elements of these collaborative applications (the Master and Participant clients) as *Collaboration Agents* in Peer-to-Peer Grid [12, 13], or preferably, *Collaboration Entities*. The elements in all the collaborative applications are just different types of entities.

We can model the entities of a type in a collaboration session using finite automata; these entities in the session are finite automaton-based; in essence, they are just finite automata, or deterministic finite automata.

It is viable of this modeling.

First, a collaboration session is finite, because human life is finite and we are only interested in modeling those adequate and meaningful sessions that are finite in time and started and ended normally. Therefore, the events invoked by a user's interaction with the interfaces on the Master client and then the event messages communicated between the Master and Participants are finite.

Second, the interfaces and widgets of an interactive windows application are finite. The event messages from some of them are the same from invocation to invocation, such as a button widget titled "Next," while others are dynamic depending on the interactive inputs, such as a text field widget. Even though in this case the event messages are different in all the widget's invocations, the invocations in a collaboration session are finite and so the associated event messages.

Third, since all our collaborative applications are designed to collaborate on events, we are only interested in the events that actually happened in a collaboration session, and model the process of the session on those events. Those events are finite. The occurrence and sequence of events in a session may be different from that in another session, and so the associated modeling finite automata.

The meaning of this modeling is that, we can get a simple, clear and consistent picture with regard to the collaboration between the entities in a session; we can see through the differences between the entities and logically abstract them to share a common finite automaton in their instantiations in collaboration; we can see the important roles of events and the shared event model in collaboration.

The Master and Participant collaboration entities are designed for different purposes, in different architectures, implementing mechanisms, and shapes of codes; they are divergent. At the same time, they have the same logic as to the state transitions on events, and get to the same state at the end of the process of each event; they are convergent.

Intuitively, the entities in collaboration – the Master and Participants – collaborate on events to keep showing the same output displays at each step, with the Master in controlling by capturing events, generating and sending out event messages to Participants through a message broker, and the Participants in responding by rendering the received event messages.

Specifically, they collaborate to share a common finite automaton in their respective instantiations – i.e., the finite automata in them are the same – and reach a common specific state of the set of states of the finite automaton at any collaboration step. Collaboration of the entities is therefore all about being in a same state of the finite automaton at each event.

Let us first describe the finite automaton.

2. THE FINITE AUTOMATON

There are two types of Finite Automaton – the Deterministic Finite Automaton (DFA) and the Nondeterministic Finite Automaton (NFA) [14]. Both of them can be represented as a five-tuple notation, as in:

$A = (Q, \Sigma, \delta, q_0, F)$, where

A is the name of the automaton;
Q is the finite set of states of the automaton;
Σ is the finite set of input symbols;
q_0 is the start state;
F is the finite set of final or accepting states, which is a subset of Q;
δ is the transition function, which takes as parameters a state from Q and a symbol from Σ, and returns a state in Q for DFA, or a set of states from Q for NFA.

The difference between a DFA and an NFA is that, for a DFA, it is in a single state at any time, while for an NFA, it has the power to be in multiple states at a time. NFA is more succinct and easier to design, while DFA is more feasible and safe in implementation and programming. Any NFA can be converted to a DFA using subset construction (the power set of the set of states of the NFA), and the two are mathematically equal. Even though in the worst case the number of states of the DFA constructed from an NFA is exponentially larger than that of the NFA (2^n vs. n), in most cases and practically, the numbers are almost equal, because most of the states of the constructed DFA are inaccessible or unreachable from the start state, and therefore can be eliminated. A useful technique in doing so is the one called lazy evaluation, which is effective in keeping all those accessible or reachable states in the power set.

Deterministic Finite Automata are suitable for each case of our projects in collaborative PowerPoint, Impress and ReviewPlus applications; the collaboration entities in them are in essence Deterministic Finite Automata.

We show the characteristics and demonstrations of the cases in the following sections.

3. CHARACTERISTICS OF THE DFA FOR COLLABORATION ENTITIES

There are characteristics in the Deterministic Finite Automata of the collaboration entities in our projects. They are the specialties in the set of input symbols Σ, and hence the transition function δ.

Traditionally, the symbols in Σ are alphabets, digits, or any printable ASCII characters; the transition function δ takes as parameters a state q_i in Q and a single symbol s_i in Σ, such as a, b, c, 1, 2, 3, %, $, &, etc., and returns a state in Q.

Specifically, we define in our cases the symbols or units in Σ to be event messages, which are independent text strings such as "OpenFile;*Dir/filename*", "Goto;*CertainSlide#*", "Previous", "Next", etc. for collaborative PowerPoint and Impress applications, and "{Widget_Base;ID:10;TOP:8;HANDLER:10;X:123;Y:456}" (representing the base widget event structure) etc. for collaborative ReviewPlus IDL applications. Each such message is defined as a "symbol" in Σ. The transition function δ takes a state in Q and such a symbol in Σ as input, and transits to the next state in Q, usually a different one.

By doing so, we encapsulate the low level chores – such as capturing events and getting the event messages, serializing them in text strings for transmitting (on the Master side), de-serializing and parsing the message strings, and building up the event structures (on the Participant side) – into the collaboration entities, and thus simplify the modeling and make it clear that the Deterministic Finite Automata in them are all about collaborations on event messages. This is to use semantically complete event messages as the basic units in the set of input symbols Σ, and make the automata concentrate on describing the message-based collaborations.

4. UNIFICATION OF THE COLLABORATION ENTITIES

If we observe the collaboration entities in a project – the Master and Participant clients – on lower levels (e.g. design and implementation), they are different things, with respect to strategies, architectures, languages and technologies used, and roles supposed. However, if we consider the entities on higher levels as to state transitions of DFA in question, they share the same state of logic at any step in a collaboration session, and therefore in essence, they have the same DFA in their respective instantiations and collaborate on it using event as the messenger.

In practice, the entities of the Master and Participant are created for different purposes; they are binary. In theory, they follow the same logic of collaboration and

manage to share the same state of a common DFA at a step in a session; they are unity.

They are binary so that they serve and satisfy the special requirements as to the capturing of events in the Master and rendering of events in the Participant. They are unity so that they have the same logic state at any collaboration step in the form of the same output screen.

In more detail, on the entity of the Master client, the user controls the process of a session by physically controlling the interfaces and widgets of the entity using mouse clicks, keystrokes, etc., which we can call physical events. The entity responds to these physical events and navigates through each of the corresponding states; at the same time for each of these events, it builds up an event message regarding information about the event such as the function to call, property or event structure, etc. The event message is a delimited text string and the intermediate representation of the event for transmission / broadcasting via the message broker.

On the entity of the Participant client, it parses the delimited text string [15] after receiving it; based on the information, it arranges which function to call, converts all the types of data represented in string to its system's interior representation, and builds up the native event structure. It then automates through each of the states as in the Master by calling a function, mostly with a property or an event structure as the parameter. It is controlled during the process of a session programmatically, which is called automation.

Considered logically, both the Master and Participant entities can be modeled as a DFA in a session. They maintain the same set of states, and collaborate on event to be in a same state at any step. The transition function δ takes the current state q and an event message (for convenience, we use event message to refer to even the native event representation of a system) as parameters and transits to the next state p of the DFA.

In Object-oriented Programming languages like C++, polymorphism is used to refer to objects of classes in different shapes, builds, and configurations yet performing the same logical functions using the same interfaces, like the "print" objects for different devices of printing hardware as well as the monitor screen.

In Peer-to-Peer Grid computing, we can use polymorphism in higher level to reference the instantiations of the collaboration entities like the Master and Participant clients – different in shapes, builds, and configurations, same in logic of the unity of the Deterministic Finite Automata. This is the Unification of the Collaboration Entities.

5. A DFA EXAMPLE IN COLLABORATIVE POWERPOINT AND IMPRESS APPLICATIONS

Let us use a DFA example suitable in collaborative PowerPoint and Impress applications to demonstrate the idea of finite automaton-based collaboration entities.

Suppose there is a presentation file either in the format of .ppt for PowerPoint or .sxi for Impress. There are three slides in this file, slide 1, 2, 3, respectively.

Accordingly, the finite set of states:

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, where

q_0 is the state when the application is started;
q_1 is the state for slide 1;
q_2 is the state for slide 2;
q_3 is the state for slide 3;
q_4 is the state when the application is ended.

The finite set of input symbols:

$\Sigma = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6\}$

Each a_i is an event message, with
a_0 = "Openfile;C:/file1.ppt" (or file1.sxi; open it),
a_1 = "Goto;1" (go to slide 1),
a_2 = "Goto;2" (go to slide 2),
a_3 = "Goto;3" (go to slide 3),
a_4 = "Exit" (the application exits),
a_5 = "Previous" (go to the previous slide),
a_6 = "Next" (go to the next slide).

The start state is q_0.

The finite set of accepting states

$F = \{q_4\}$

We define q_4 – the state when the application is exited – to be the accepting state of the automaton. That means the presentation session is normally and adequately finished.

As for the transition function δ is concerned, instead of using many equations $\delta(q_i, a_i) = p_i$ to represent the transition from state q_i to p_i on input symbol (event message) a_i, we make use of two more convenient and clearer means to do the job. They are the transition

diagram and the transition table. We will demonstrate one of them with the example next.

So, the five-tuple notation A = (Q, Σ, δ, q_0, F) for the DFA in this example becomes

A = ({q_0, q_1, q_2, q_3, q_4}, {a_0, a_1, a_2, a_3, a_4, a_5, a_6}, δ, q_0, {q_4})

Transition Diagram

Next, we give the transition diagram for the transition function δ, and explain how the transitions go through the states of Q in this example.

The transition diagram is in Figure 1.

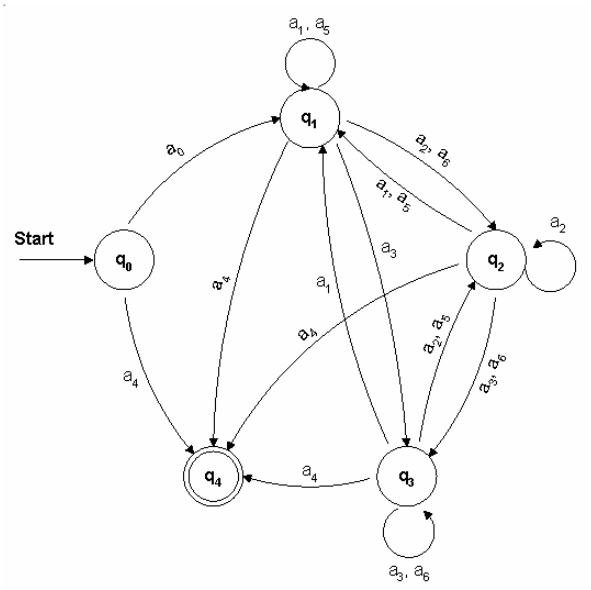

Figure 1. The Transition Diagram For The DFA Of The Collaboration Entities Working On A Presentation File In PowerPoint Or Impress Of OpenOffice

Explanation of the Example

After a collaboration entity is instantiated, it is in state q_0 which is the start state and denoted by an arrow; from here, it can either exit immediately without doing anything by going to state q_4 on message a_4 = "Exit"; or it can go to state q_1 on message a_0 = "Openfile;C:/file1.ppt", at which the presentation file is opened and by default is on slide 1.

From state q_1 it can go to state q_2 which is the state for slide 2 on message a_2 = "Goto;2" or a_6 = "Next"; or it can go to state q_3 which is the state for slide 3 on message a_3 = "Goto;3"; or it can go to state q_1 itself on message a_1 = "Goto;1" or a_5 = "Previous" (because for slide 1, there is no previous slide for it, so it stays); or it can go to state q_4 which is the state when the collaboration entity is killed on message a_4 = "Exit".

From state q_2 it can go to state q_1 which is the state for slide 1 on message a_1 = "Goto;1" or a_5 = "Previous"; or it can go to state q_3 on message a_3 = "Goto;3" or a_6 = "Next"; or it can go to state q_2 itself on message a_2 = "Goto;2"; or it can go to state q_4 on message a_4 = "Exit".

From state q_3 it can go to state q_1 on message a_1 = "Goto;1"; or it can go to state q_2 on message a_2 = "Goto;2" or a_5 = "Previous"; or it can go to state q_3 itself on message a_3 = "Goto;3" or a_6 = "Next" (because for slide 3, there is no next slide for it, so it stays); or it can go to state q_4 on message a_4 = "Exit".

State q_4 is the state when the collaboration entity is ended, and is the accepting state. Nothing will happen from here, and therefore there is no label leading out from it. It is denoted by a double circle.

In this example we have discussed all flow possibilities for a 3-slide presentation. It implies many possible collaboration sessions. For example, in one session slides 1, 2, and 3 are presented in that order and then the session is ended; in another session the slides may be presented randomly and in any number. In the modeling of one actual collaboration session, the corresponding deterministic finite automaton is a sub-graph of the transition diagram of Figure 1.

6. ISSUES ABOUT DFA WITH COLLABORATIVE REVIEWPLUS APPLICATIONS

From a point of view, collaboration of the Collaborative ReviewPlus applications is essentially all about the synchronization of the interfaces between the Master and Participant clients at each step.

In IDL, the interface consists of all kinds of widgets such as buttons, lists, sliders, tabs, text fields, etc. The constitution, configuration and layout of the widgets in the interfaces of an application are coded in its widget programs.

The states of the DFA in question are based on the widgets. The relationships between the widgets and the states may be one-to-one correspondence, or one-to-many.

Examples of the former are: 1) a simple button widget – one button click causes the DFA to transit to the next state; 2) a text field widget configured with the keyword "return_events" in the widget program, which means that an event is only fired when the carriage return key is pressed in the text field. Any string content typed in the field is reflected in the event structure as a single string value after the pressing of the return key, and the DFA transits to the next state with this string value.

An example of the latter is a text field widget configured with the keyword "all_events", which means an event is fired whenever the contents of the text field have changed. Each character input in this case triggers an event, including the ending hexadecimals "0a" and "0d" for line feed and carriage return, if there were some. Such a widget corresponds to the transition through one or more states when finished.

This feature of "all_events" for a text field widget makes it possible to show more detailed process in collaboration, as opposed to the case with the feature "return_events" where only the final value in the field is communicated, without showing the detailed actions of the inputs in the participant clients.

7. EXTENDED TRANSITION FUNCTION WITH COLLABORATION ENTITIES

The type of collaboration entities we have described so far is synchronous; that is, the Master and Participants are cooperating in the same session and sharing the same output displays in real time.

The other type of collaboration entities we yet have to describe is of asynchronous – the information and data about the collaboration during a synchronous session is recorded and saved, and the asynchronous entities can access them at any time thereafter, in any possible way, taking advantage of the *Extended Transition Function* Δ [14].

More specifically, we can make the synchronous entities save the event messages a_i in a session in the order they happened, and connect them in a string ω, as in

$$\omega = a_0 a_1 \ldots a_n$$

Later on in the asynchronous access, the Extended Transition Function Δ makes use of any prefix of the string ω, e.g. $a_0 a_1 \ldots a_i$ (with $0 \le i \le n$), and transits to state q_{i+1}. This means that the users with the asynchronous entities can review the content happened in a session in a way that is sequential access, random access, or even "keyword search" based access to any history display of the contents – the concept of *Reverse Indexing on Event Messages*. We shall address the idea step by step in the following sub-sections.

7.1. Extended Transition Function

The Extended Transition Function Δ is a function that takes a state q and a string ω, and returns a state p, as in

$$\Delta (q, \omega) = p$$

The automaton starts at state q, processes the sequence of string ω, and finally reaches state p.

It is defined by induction on the length of the string ω, as follows.

BASIS: $\Delta (q, \varepsilon) = q$. That is, if the automaton is in state q and reads no input or a null string, it is still in state q.

INDUCTION: Suppose $\omega = a_0 a_1 \ldots a_n$, $x = a_0 a_1 \ldots a_{n-1}$, $a = a_n$, we can write $\omega = xa$, in which "a" is the last symbol of the string ω, and "x" is the rest of it. Then,

$$\Delta (q, \omega) = \delta (\Delta (q, x), a)$$

The Extended Transition Function Δ is based on the Transition Function δ. Let

$$\Delta (q, x) = p$$

Then
$$\Delta (q, \omega) = \delta (\Delta (q, x), a) = \delta (p, a)$$

That is, for any length of string x, if the final state is p due to the transitions on the sequence of x, then the next state on one more input symbol a is decided by the transition function δ, as in $\delta (p, a) = r$.

7.2. The Language Of A DFA

We have defined the symbols of Σ to be event messages. In a collaboration session between the entities, the actual event messages are finite, which is mainly decided by the finiteness of the session. Let

$$\Sigma = \{a_0, a_1, a_2, \ldots, a_n\}$$

All the event messages a_i in Σ could have formed random strings in any length, any combinations of the a_i's, in any order. Examples are: ε, a_0, a_1, a_2, $a_0 a_0$, $a_1 a_1 a_1$, $a_0 a_1 a_2$, $a_0 a_1 a_2 \ldots a_n$, $a_3 a_n a_2 \ldots a_0$, and the like. We denote the set of all strings constructed from the symbols in Σ to be Σ^*.

Not all the strings in Σ^* are possible or meaningful for a collaboration session. We are only interested in those strings that cause the DFA to go through transitions from the start state q_0 to an accepting state in F, such as $\omega = a_0a_1 \ldots a_n$, and when we refer to a collaboration session we mean such a successful one that leads to an accepting state.

One collaboration session is associated with one event message string $\omega = a_0a_1 \ldots a_n$, and all such strings form a *Language* for a type of collaboration entities – Collaborative PowerPoint, Collaborative Impress, Collaborative ReviewPlus, or others.

If the DFA for the type of collaboration entities is $A = (Q, \Sigma, \delta, q_0, F)$, then the language L (A) is defined as

$$L(A) = \{\omega \mid \Delta(q_0, \omega) \in F\}$$

That is, the set of strings which cause the DFA to go through transitions from the start state q_0 to an accepting state in F.

7.3. Random And Sequential Access

In random access of an asynchronous session, the user directs the entity to randomly go to an event message a_i $(0 \leq i \leq n)$ in string $\omega = a_0a_1 \ldots a_n$, generate the corresponding state p and render the output display. The entity does the job by taking advantage of the Extended Transition Function Δ, as in

$$\Delta(q_0, x) = p, \text{ where } x = a_0a_1 \ldots a_i.$$

The entity basically begins with the start state q_0, goes through all the transitions in response to each a_j $(0 \leq j \leq i)$ in x and finally gets to state p on input symbol a_i.

In sequential access, the Extended Transition Function Δ can be used in the same way as in random access, but since in sequential access the symbols in string $\omega = a_0a_1 \ldots a_n$ are accessed sequentially one by one from a_0 going forward to a_n, the entity can just take advantage of the current state p in memory, get the next symbol a_j $(0 \leq j \leq n)$ in the remaining string $a_ja_{j+1} \ldots a_n$ of ω, and go to the next state r by the transition of the Transition Function δ, as in $\delta(p, a_j) = r$. The basis of Δ is δ, any way.

7.4. Reverse Indexing On Event Messages

The Web Browsers nowadays have keyword search mechanisms to find relative web sites based on the input keywords and list them for the user to click on. One of the most popular search engines is Google. The technique they use is the one that is called "Reverse Index" – keywords associate with web sites.

We can make use of this technique on the event messages a_i of string ω in an asynchronous session.

Because an event message a_i in this case corresponds to a state q, which in turn corresponds to an output display, which corresponds to some contents, from which keywords can be generated, therefore, we can associate the keywords with the event message a_i and hence this becomes *Reverse Indexing on Event Messages*.

Later on, the user with the asynchronous collaboration entity can use keywords to get event messages a_i and then $x = a_0a_1 \ldots a_i$, and use $\Delta(q_0, x) = p$ to get to the states and therefore find the contents.

Further more, this can bring different languages into collaboration. Let us use

$$\omega_1 = a_0a_1 \ldots a_k$$

to denote the strings in the language of the entities of the Collaborative PowerPoint;

$$\omega_2 = b_0b_1 \ldots b_m$$

to denote the strings in the language of the entities of the Collaborative Impress; and

$$\omega_3 = c_0c_1 \ldots c_n$$

to denote the strings in the language of the entities of the Collaborative ReviewPlus.

They are different languages and for different purposes of usage. PowerPoint and Impress are designed mainly for the presentation of text, while Reviewplus for graphics and images, 2D or 3D.

Suppose a lecture was presented using all the three types of the above collaboration entities, and the event messages were saved in all the three languages, and the keywords associated with the event messages for the related contents are consistent. Then the user in an asynchronous session can use keyword search to find the text representations of the contents in both the asynchronous entities of the Collaborative PowerPoint and the Collaborative Impress, and find the graphic/image representations in the asynchronous entity of the Collaborative ReviewPlus.

8. LOGICAL CONSENSUS

In this section, we use the Collaborative ReviewPlus as an example to describe some issues of the collaborative applications, mainly focusing on event and logic with the applications. We shall see that the Master and Participant clients share a common Deterministic Finite Automaton (DFA) in a session, have the same logic with regard to the state transitions, and converge on a same state on each event.

The same holds for Collaborative PowerPoint, Collaborative Impress, and other such collaborative applications.

We describe the issues as follows.

8.1. Units And Unity

We have developed the Collaborative ReviewPlus applications – the Master and Participant collaboration entities – from the original ReviewPlus application, without changing the overall logic related to state transitions. So they have the same logic with regard to the state transitions on events.

The logic corresponds to the transition function δ of the DFA, or the extended transition function Δ. The logic is composed of many IDL routines – procedures and functions with unique names. We can think of the routines as the building blocks or *units* of the logic, and the logic as the *unity* of the routines. So, routines are the units, and δ or Δ is the unity of the units.

On an event, only one or some routines are executing to do the transition; in other words, only one part or some parts of the unity are actually functioning. But we can indistinguishably say that δ or Δ is reacting on the event and transiting to the next state.

8.2. Divergence And Convergence

The Master and Participant collaboration entities are designed for different purposes, in different architectures, implementing mechanisms, and shapes of codes; they are divergent. At the same time, they have the same logic as to the state transitions on events, and get to the same state at the end of the process of each event; they are convergent.

Let us describe it in more detail using the implementation of the Collaborative ReviewPlus applications as the example. It is similar for the others.

On the Master client, each widget that fires event is associated with an event handler – either a procedure or a function – in the widget construction programs, which are registered at the end of the constructions with the IDL system routine "xmanager.pro", which in turn is managing the life-cycle of the widgets and listening for events from them. Whenever a widget is triggered by the user through the interface, the system automatically gathers the information for the event and fits in the event structure, and invokes the event handler with the event structure as the only parameter.

We add the code for collaboration here at the beginning of each event handler to capture the event and get the information of it for every field of the event structure, convert them into flat strings, serialize them into a delimited single string along with names of the event structure and the event handler, and send this result string to NB message broker for broadcasting to participant clients.

NB broadcasts the string to the participant and saves it in a public variable which is one element of a synchronized linked list added in one of NB's interface class, and also updates its event flag variable which reflects the number of strings saved.

The Participant client is developed using a Polling Structure. It is a main loop that is constantly polling the public variables – testing the event flag to see if it is non-zero; if it is, then removing a string from the head of the linked list to do further process.

In the process, it parses the string on the delimiter [15] to get all the field pieces, the event structure name (or widget name) and the event handler name, converts the field pieces to native type values of the event structure, constructs the event structure using these values according to the event structure name, and finally renders the display by calling the event handler routine with the event structure as parameter, according to the event handler name.

As to the interactive input value in an input field such as text field, on the Master side, the user input them physically; on the Participant side, after it gets the value, it sets it in the field programmatically.

From the description above, we can see that the entities of the Master and Participant clients diverge in the shapes of codes, architectures, implementing mechanisms and purposes. They are in diversity under the goal of collaboration.

However, on each event, we have made them have the same input value in the input field if there is one, call the same routine of event handler with the same event structure as parameter, and hence, at the end of the

processes of the event, have the same output display; in other words, they converge on the same state of the DFA on each event, from the start state to the final accepting state, which is a well-defined session.

8.3. Collaboration On Event And Transition Function

We now demonstrate the collaboration between the entities of the Master and Participant clients using pieces of code from the Collaborative ReviewPlus applications, mainly focusing on event and the transition function. We just give one collaboration step here that illustrates the idea of collaboration in terms of convergence on the same state of the DFA at the end of the process of the event, with the transition function doing the real job of state transition. In our technical report, *A Description of the Implementation of Collaborative ReviewPlus* [16], we gave more such typical and interesting ones that would sufficiently help to get the idea.

We can see from the one step collaboration demonstration that, the Master and Participant collaboration entities are designed for different purposes, in different implementations and shapes of codes; they are divergent. At the same time, they have the same logic as to the state transitions on the event of the step, and get to the same state at the end of the process of the event; they are convergent.

Since the output displays of both the Master and Participant clients at the event are the same, we just show a single set of image captures in the demonstration of the step.

We begin with the invocation of the collaboration entities, as shown in Figure 2. This corresponds to the start state q_0 of the DFA.

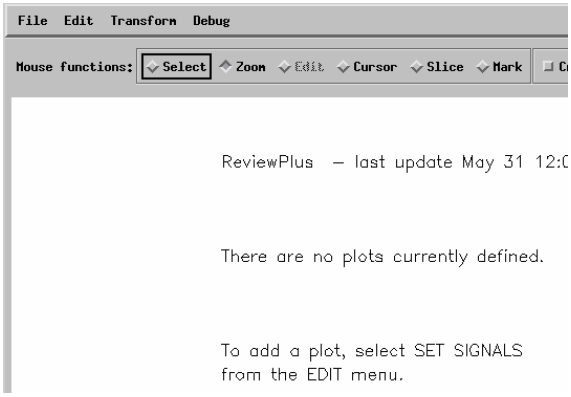

Figure 2. A Part Of The Initial Interface And Display Of ReviewPlus

From this interface on the Master client, if we click on the "Edit" item from the main menu, a sub-menu will appear, as shown in Figure 3.

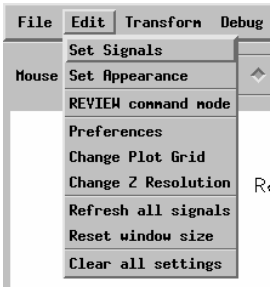

Figure 3. A Sub-menu From The Main Menu Of ReviewPlus

If we then click on the "Set Signals" item from the sub-menu, an event is fired. This is a button widget, and an event handler routine is defined for the event. We describe the pieces of code for both the Master and Participant clients in achieving collaboration in response to this event as follows.

The Master Client Side

- Widget creation

```
  x = widget_button(mEdit, value='Set
Signals', $
event_pro='ReviewPlus_SignalDialog_event')
```

From the code above we know that this button widget has `'Set Signals'` as its value shown on its appearance, and is associated with an event procedure named `'ReviewPlus_SignalDialog_event'`. When the button is clicked, the procedure is called by the IDL system.

- Definition of event structure for widget

Here is the definition of the event structure for widget button:

{WIDGET_BUTTON, ID:0L, TOP:0L, HANDLER:0L, SELECT:0L}

It has a name WIDGET_BUTTON and 4 fields – ID:0L, TOP:0L, HANDLER:0L, and SELECT:0L, each with a field name, a colon, and a type value. In this case all the values of the fields are of `long` type indicated by the suffix letter L.

SELECT: If the button is pressed, the value is 1; if it is released, the value is 0.

- Event handler

```
pro ReviewPlus_signaldialog_event,event
;;; collaboration code added ;;;
eventMessage =
"ReviewPlus_signaldialog_event;"+"WIDGET_BU
TTON;"+"ID;"$
+string(event.ID)+";TOP;"+string(event.TOP)
+";HANDLER;"$
+string(event.HANDLER)+";SELECT;"+string(ev
ent.SELECT)
  COMMON BROKER, joChat2
  joChat2 -> writeMessage, eventMessage
;;; end of collaboration code ;;;
widget_control,event.top,get_uvalue=info
  info.oReview->SignalDialog
end
```

From the code above we can see that the collaboration code captures the event and gets its field information from `event.ID`, `event.TOP`, `event.HANDLER`, etc., converts them into strings and serializes the strings into a semicolon delimited string, along with the event structure name `WIDGET_BUTTON` and the event handler name `ReviewPlus_signaldialog_event`. This result string is the event message, and is sent to the NB broker for broadcasting to Participants.

The Participant Client Side

- Parsing of event message

```
          result = STRSPLIT(uval, ';',
COUNT=count, /EXTRACT, /PRESERVE_NULL)
          which_event = result[0]
          which_widget = result[1]
```

The next event message string for the Participant client to process is saved in variable `uval`. The IDL system function `STRSPLIT` is called to parse it with `';'` as the delimiter. All the pieces of information around the delimiter are extracted and saved in the array `result` with null string preserved as a piece, and the total number of them is saved in variable `count`. The event handler name is in `result[0]` or `which_event`, and the event structure name (or widget name) is in `result[1]` or `which_widget`. The rest of the pieces are all for the fields of the event structure and are saved in the rest elements of the array starting with `result[2]`.

- Conversion to IDL native types

```
FOR i=2, count-1, 2 DO BEGIN
    IF (result[i] EQ 'ID') THEN BEGIN
        id_name = 'ID'
        id_value = long(result[i+1])
    ENDIF ELSE IF (result[i] EQ 'TOP')
    THEN BEGIN
        top_name = 'TOP'
        top_value = long(result[i+1])
    ENDIF ELSE IF (result[i] EQ
                          'HANDLER')
    THEN BEGIN
        handler_name = 'HANDLER'
        handler_value = long(result[i+1])
    ENDIF ELSE IF (result[i] EQ 'SELECT')
    THEN BEGIN
        select_name = 'SELECT'
        select_value = long(result[i+1])
                    :
    ENDIF
ENDFOR
```

The code above converts the information (in string) of the fields of the button event structure to its IDL native types; each pair of the strings, i.e. those stored in `result[i]` and `result[i+1]`, decide the field's value and the type of the value, with the former indicating the name and type of the value (due to the unique association of a name with a type, the name alone can also indicate a type, e.g. `ID` is a `long` type), and the latter the value in string.

In this case, all the values of the fields are of `long` type, and therefore the strings are converted to IDL type `long`.

- Construction of event structure

```
IF (which_widget EQ 'WIDGET_BUTTON') THEN
event_structure =
{WIDGET_BUTTON,id:id_value,$
top:top_value,handler:handler_value,select:
select_value}$
          ELSE IF ...
```

The code above constructs the widget button event structure using the converted native values for each field, with the field name followed by a colon and then by the value, as in `id:id_value`.

- Invocation of the routine of event handler

```
   ...
   ELSE IF (which_event EQ
'ReviewPlus_signaldialog_event') THEN BEGIN
     ReviewPlus_signaldialog_event,
event_structure
        ENDIF ELSE IF ...
```

The code above calls the routine of the event handler `ReviewPlus_signaldialog_event` with the constructed event structure `event_structure` as the only parameter.

Step Summary

In the process on the event, both the Master and Participant clients call the same routine – the event handler `ReviewPlus_signaldialog_event` – which is a unit of the transition function δ, with the event structure as the only parameter. The event message acts as the messenger, the information source, and the coordinator.

With $\delta(q_0, a_0) = q_1$, the Master and Participant clients converge on the same state q_1 of the DFA on event message a_0 at the end of the process of the event, and therefore they have the same output display, as in Figure 4, which is a part of a big interface.

Note that, inside an event handler, other routines can be called in any sequence and order, which we do not have to worry about but just think of the whole as the encapsulation and abstraction of the event handler.

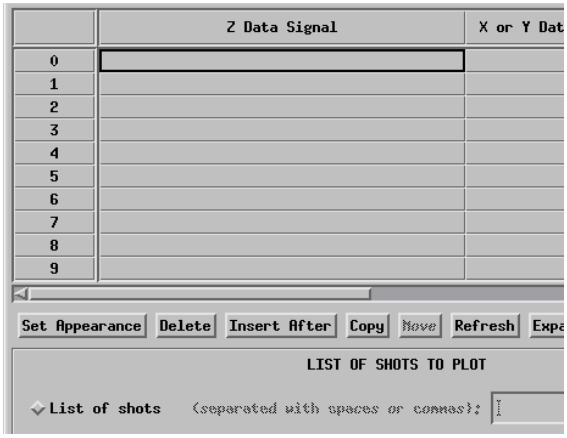

Figure 4. A Part Of A Big Interface In ReviewPlus For Setting And Managing Of Signals

9. CONCLUSION

In this paper we introduce several types of collaborative applications that use the Shared Event Model in Peer-to-Peer Grid computing.

Each application type consists of collaboration entities, and they play different roles in collaboration.

We model the entities in a collaboration session to be finite automaton-based, and point out that they are deterministic finite automata in essence.

We describe the characteristics of the automata in modeling and analysis of the collaboration between the entities. We discuss issues in both synchronous and asynchronous collaborations.

We demonstrate that the collaboration entities of a type converge on a common state of the deterministic finite automata at a collaboration step, even though they diverge in many other respects.

Intuitively the entities collaborate on events to keep showing the same output displays; specifically collaboration of the entities is all about being in a same state of the deterministic finite automata at each event.

REFERENCES

[1] Minjun Wang, Geoffrey Fox, and Shrideep Pallickara, "Demonstrations of Collaborative Web Services and Peer-to-Peer Grids," *Journal of Digital Information Management*, Digital Information Research Foundation, Volume 2, Issue 2, June 2004, pp. 93-96.
http://grids.ucs.indiana.edu/ptliupages/publications/P2PGrids_JDIM_PDF.pdf

[2] Minjun Wang and Geoffrey Fox, "Design of a Collaborative System," Proceedings of the IASTED International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2004), ACTA Press, St. Thomas, US Virgin Islands, November 22-24, 2004, pp. 192-200.
http://grids.ucs.indiana.edu/ptliupages/publications/OpenOfficeCollaborativeSystemFINAL.pdf

[3] Minjun Wang, Geoffrey Fox, and Marlon Pierce, "Grid-based Collaboration in Interactive Data Language Applications," Proceedings of ITCC 2005 International Conference on Information Technology: Coding and Computing, IEEE Computer Society, Las Vegas, Nevada, USA, April 4-6, 2005, Volume I, pp. 335-341.
http://grids.ucs.indiana.edu/ptliupages/publications/GridCollabIDL_ITCC2005.pdf

[4] Minjun Wang, Geoffrey Fox, and Marlon Pierce, "Instantiations of Shared Event Model in Grid-based Collaboration," Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2005), International Institute of Informatics and Systemics, Orlando, Florida, USA, July 10-13, 2005, Volume III, pp. 11-18.
http://grids.ucs.indiana.edu/ptliupages/publications/InstantiationsGridCollab.pdf

[5] Geoffrey Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, and Wenjun Wu, "Collaborative Web Services and Peer-to-Peer Grids," Proceedings of 2003 Collaborative Technologies Symposium.
http://grids.ucs.indiana.edu/ptliupages/publications/foxwmc03keynote.pdf

[6] Fran Berman, Geoffrey Fox, and Tony Hey (editors), GRID COMPUTING: MAKING THE GLOBAL INFRASTRUCTURE A REALITY, John Wiley & Sons Ltd, Chichester, West Sussex PO19 8SQ, England, 2003.

[7] ReviewPlus Data Visualization Software User Manual
http://web.gat.com/comp/analysis/uwpc/reviewplus/manual/

[8] OpenOffice.org
http://www.openoffice.org/

[9] Virtual Network Computing
http://www.realvnc.com/

[10] Geoffrey Fox, Shrideep Pallickara, and Xi Rao, "A Scalable Event Infrastructure for Peer to Peer Grids," Proceedings of 2002 Java Grande/ISCOPE Conference, ACM Press, Seattle, November 2002, pp. 66-75.
http://grids.ucs.indiana.edu/ptliupages/publications/ScaleableEventArchForP2P.doc

[11] Shrideep Pallickara and Geoffrey Fox, "Efficient Matching of Events in Distributed Middleware Systems," *Journal of Digital Information Management*, Digital Information Research Foundation, Volume 2, Issue 2, June 2004, pp. 79-87.
http://grids.ucs.indiana.edu/ptliupages/publications/jdim-vol2-num2.pdf

[12] Andy Oram (editor), PEER-TO-PEER: HARNESSING THE POWER OF DISRUPTIVE TECHNOLOGIES, O'Reilly & Associates, Inc., Sebastopol, CA 95472, USA, 2001.

[13] Ian Foster and Carl Kesselman (editors), THE GRID: BLUEPRINT FOR A NEW COMPUTING INFRASTRUCTURE, Morgan Kaufmann Publishers, Inc., San Francisco, CA 94104-3205, USA, 1999.

[14] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, INTRODUCTION TO AUTOMATA THEORY, LANGUAGES, AND COMPUTATION, Addison-Wesley, Boston, MA, USA, 2001.

[15] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, COMPILERS: PRINCIPLES, TECHNIQUES, AND TOOLS, Addison-Wesley Publishing Company, USA, 1988.

[16] Minjun Wang, "A Description of the Implementation of Collaborative ReviewPlus," Technical Report, July 19, 2005.
http://grids.ucs.indiana.edu/ptliupages/publications/Generalization_ReviewPlus.pdf